VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Reimplementace utilit pro systém Virtlab spolu s implementací webového rozhraní pro správu rezervacního serveru

Reimplementation of System Utilities, Implementation of Web Interface for Reservation Management of Virtlab System

2012 Roman Petráš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Roman Petráš**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Reimplementace utilit pro systém Virtlab spolu s implementací webového rozhraní pro správu rezervačního serveru
Reimplementation of System Utilities, Implementation of Web Interface for Reservation Management of Virtlab System

Zásady pro vypracování:

Cílem práce je reimplementovat jednotlivé systémové a administrátorské utility v rámci stávajícího řešení systému Virtlab. Jedná se o modul pro hlášení chyb, modul pro sledování uživatelů a graf zátěže na laboratorním vybavení. Dále bude navržen a následně implementován nový modul pro správu rezervačního serveru přes webové rozhraní. Výsledné řešení bude integrováno do nové implementace systému Virtlab.

1. Seznamte se s jednotlivými původními moduly, převážně pak s implementací rezervačního serveru.
2. Seznamte se s novou implementací systému Virtlab, která je implementována pomocí frameworku Nette.
3. Navrhněte nový modul pro správu rezervačního serveru přes webové rozhraní systému Virtlab.
4. Navržené řešení včetně přepracovaného původního řešení implementujte pomocí frameworku Nette.
5. Funkčnost jednotlivých subsystémů řádně otestujte a zdokumentujte.

Seznam doporučené odborné literatury:

NOVÁK, Radek. Robustní reimplementace prototypového řešení serverových komponent systému Virtlab. Ostrava, 2010. 72 s. Diplomová práce. VŠB-TU Ostrava, FEI.
GRAMES, Martin. Reimplementace řídící aplikace systému Virtlab s použitím moderních webových technologií. Ostrava, 2010. 47 s. Diplomová práce. VŠB-TU Ostrava, FEI.
Stránky projektu Virtlab [online]. 2006-06-28 [cit. 2010-11-05]. Virtuální laboratoř počítačových sítí. Dostupné z WWW: <http://www.virtlab.cz>.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Kateřina Bambušková**

Datum zadání: 19.11.2010
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 3.5.2012 .....

.....
Roman Petráš

Poděkování

Tímto bych chtěl poděkovat Ing. Kateřině Bambuškové, vedoucí bakalářské práce, za všechny podněty, vstřícnost a ochotu při vypracování této bakalářské práce.

Abstrakt

Tato bakalářská práce se zabývá reimplementaci a zároveň rozšířením webové řídící aplikace distribuované virtuální laboratoře počítačových sítí. Zahrnuje vysvětlení a postup k používání implementovaných doplňků do nového systému. Dále obsahuje popis reimplementace jednotlivých utilit do nového systému využívajícího framework Nette pro jazyk PHP. A jako hlavní část práce je zanalyzovat, navrhnout řešení a implementovat do nového systému Virtlab uživatelské rozhraní pro správu rezervačního serveru přístupnou administrátorům Virtlabu.

Klíčová slova:

Virtlab, Nette, PHP

Abstract

This bachelor work deals with reimplementation and extension of the web managing application in distributed virtual laboratory of the computer networks. This work includes explanation and approach of using implemented add-ins to the new system. The next it contains the description of reimplementation individual utilities to the new system using Nette framework for PHP language. As the main part of the work is analyze, devise solution and implement user interface for administration of the reservation system which is accessible for Virtlab administrators to the new Virtlab system.

Keywords:

Virtlab, Nette, PHP

Seznam použitých zkratek a symbolu

PHP – Personal Home Page – Hypertext Preprocesor – skriptovací jazyk

Virtlab – Virtuální laboratoř počítačových sítí

VirtIS – Informační systém Virtlabu

XML – Extensible Markup Language – Obecný značkovací jazyk

HTTP – Hypertext Transfer Protocol - internetový protokol určený pro výměnu hypertextových dokumentů ve formátu HTML.

SEO – Search Engine Optimization - metodika vytváření a upravování webových stránek

AJAX - Asynchronous JavaScript and XML - technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich znovunačítání.

Obsah

1 Úvod

Virtlab neboli distribuovaná virtuální laboratoř, je prostředníkem mezi reálnými síťovými prvky laboratoře a webovým rozhraním. Od prvního nápadu o vytvoření virtuálního přístupu k síťovým prvkům někdy v roce 2005, prošel systém dlouhým vývojem. Dnešní verze se od původní zásadně liší. Je to způsobené především týmem vývojářů, kteří se podílí na udržování a vývoji tohoto systému. Hlavní myšlenka ovšem zůstává stejná.

V dnešní době je v provozu distribuovaná verze Virtlabu, která má pomoct především při výuce zaměřené na počítačové sítě. Studenti tak mají možnost pracovat na, poměrně drahých, cenově nedostupných síťových prvcích laboratoře a to vzdáleně přes internet. To jim umožňuje také rezervační server, který slouží pro rezervaci jednotlivých zařízení. Umožňuje tak pracovat více studentům najednou, což je ovšem omezeno počtem síťových prvků. Studenti si tak mohou rezervovat úkoly nebo vlastní topologie na určitý čas, o zbytek se postará rezervační server.

Cílem bakalářské práce je vytvořit administrátorské prostředí pro správu tohoto rezervačního serveru. Tak aby bylo možné, zpravovat různé funkce, bez toho aniž by musel administrátor zasahovat ručně do databáze, jak tomu bylo dříve.

Další část práce pokračuje v robusní reimplementaci celého systému do nové podoby. Od nového systému se očekává hlavně lepší funkcionalita a vymezení chyb, které se vyskytovaly na původní verzi systému. Konkrétně se práce týká modulu pro hlášení chyb, grafu zátěže a sledování uživatelů.

2 Nové doplňky pro framework Nette

Okolo frameworku Nette je rozsáhlá komunita programátorů, kteří vyvíjejí pro tento framework různé doplňky nebo rozšíření a tak ulehčují ostatním programátorům práci. Všechny doplňky jsou dostupné na oficiálních stánkách frameworku. Pro potřeby bakalářské práce byly také využity tyto výhody a zakomponovány do systému 2 doplňky *DataTimePicker* a možnost práce s AJAXovým rozšířením.

2.1 DataTimePicker

Jelikož ve větší části této práce je potřeba použít formuláře, ve kterých se objevují časové hodnoty, rozhodl jsem se zakomponovat do projektu *DataTimePicker*. Je to předpřipravený graficky zpracovaný doplněk pro framework Nette. Slouží právě pro zadávání těchto časových údajů do formulářů. Výhodou jsou validní výstupní data a již zmíněné grafické zpracování, kdy uživatel má možnost zadat datum a čas pouhým kliknutím na zobrazený kalendář.

Obrázek 1: Náhled DataTimePicker formuláře

Prvek je zahrnut pro třídy *Form* a *AppForm* z důvodu možnosti pozdějšího využití. Pro nastavení *DataTimePickeru* jsou vytvořeny dvě funkce. Ty jsou implementovány v souboru *bootstrap.php*.

```
function Form_addDateTimePicker(Form $_this,$name,$label,$cols=NULL,$maxLength = NULL) {
       return $_this[$name] = new DateTimePicker($label, $cols, $maxLength);
}
function  AppForm_addDateTimePicker(AppForm  $_this,$name,$label,$cols=NULL,$maxLength  =
NULL){
       return $_this[$name] = new DateTimePicker($label, $cols, $maxLength);
}

Form::extensionMethod('addDateTimePicker', 'Form_addDateTimePicker');
AppForm::extensionMethod('addDateTimePicker', 'AppForm_addDateTimePicker');
```

Následné použití *DataTimePickeru* je celkem jednoduché. Spočívá ve volání funkce *addDateTimePicker v* komponentě formuláře, kde by se měl vyskytovat. Tváří se jako funkce třídy *Form* (nebo *AppForm*), proto lze také volat základní funkce, jakými jsou například *addRule* pro ošetření výstupu nebo *setDefaultValue* pro nastavení výchozí hodnoty.

```
protected function createComponentNameForm() {
      $form = new AppForm;
      $form->addDateTimePicker(name, label);
      return $form;
}
```

2.2 AJAX

AJAX umožňuje, aby stránka kontaktovala server a obdržela od něj libovolná data v XML, bez toho, aby se musela celá znovu nahrávat. Vše jen pomocí JavaScriptu. To znamená, že např. při stisku odkazu nebo tlačítka se z databáze na serveru, načtou aktuální data.

Jelikož nové weby z větší části již využívají AJAX pro paralelní odesílání a zpracování dat, tak jsou také zahrnuty do projektu. Nette je pro AJAX celkem dobře vybavené. Hlavní

důvod pro toto použití vznikl při implementaci grafu zátěže. V okamžiku odeslání formuláře, pro vykreslení grafů, AJAX spustí další funkce na pozadí. Zobrazí se jen uživatelský přívětivá ikona upozorňující načítání grafu. Ale k tomu později.

2.2.1 Použití AJAXu v Nette

V nette je použití AJAXu celkem jednoduché díky tzv. snippetům. Tyto speciální bloky kódu se přenášejí ke klientovi, který si podle nich aktualizuje stránku. Těmito bloky se obalí kus kódu v šabloně, ten se poté zpracuje za pomocí AJAXu. Bloky mohou být pojmenovány.

```
{snippet [jmeno]} kód zpracovaný AJAXem {/snippet}
```

Je potřeba také vyvolat událost, která tento blok spusti. Pro každou takovou událost ovšem bude potřeba nastavit JavaScriptové funkce. Funkce hledají třídy s názvem ajax, které pak při kliku zpracovává.

```
$("a.ajax").live("click", function (event) {
      event.preventDefault();
      $.get(this.href);
});

$('form.ajax').live('submit', function (event) {
      event.preventDefault();
      $.post(this.action, $(this).serialize());
});
```

V projektu jsou nyní nastaveny funkce pro dvě události. Již zmíněné odesílání formuláře a událost pro HTML tag <*a*>, nebo-li tzv. hypertextové odkazy.

Do projektu byly zahrnuty také funkce pro skrývání flash zpráv po 5 sekundách. Nemá to žádny vliv na funkčnost webových stránek, ale oživí to statický vzhled Virtlabu.

```
$(".flash").livequery(function () {
      $(this).delay(5000).animate({"opacity":0},
      2000).slideUp();
});
```

Poslední co je potřeba ošetřit jsou právě zmíněné flash zprávy. Při vyvolání události AJAXem se provede kód obsažen v bloku *snippet*. Proto také flash zprávy musí být obaleny tímto blokem

```
{snippet flashes}
      {foreach $flashes as $flash}
      <p class="flash msg {$flash->type}">{$flash->message}</p>
      {/foreach}
{/snippet}
```

Tak aby nebylo potřeba neustále volat funkci zobrazení flash zprávy AJAXem, tak se vkládá do *BasePresenteru.php* do funkce *afterRender()*. Z názvu funkce již vyplívá, že je volána po každém načtení šablony. To nám zajistí, že funkce se spustí vždy, kdy je potřeba.

```
public function afterRender(){
       if ($this->isAJAX() && $this->hasFlashSession())
           $this->invalidateControl('flashes');
}
```

2.2.2 AJAX pro odesílání formulářů

Pro odeslání formuláře je potřeba nastavit správně komponenta, tak aby při zapnutém JavaScriptu v prohlížeči, byla komponenta zpracována AJAXem. Formulář musí mít nastavenou třídu „ajax“ a to se v nette provede příkazem:

```
$form->getElementPrototype()->class('ajax');
```

Poté při odeslání formuláře se zjistí, zda je AJAX k dispozici. Pokud není k dispozici, formulář je zpracován přesměrováním na parametr *action.*

```
if(!$this->isAJAX())
      $this->redirect('this');
else
      $this->invalidateControl('form');
```

Parametr funkce *invalidateControl* slouží k označení snippetu, který na tuto událost má reagovat. Ty mohou být pojmenovány, tak aby bylo možné, využit v jedné šabloně více takových funkcí.

2.2.3 AJAX pro odeslání odkazu

Nastavení je v podstatě stejné jako při odesílání formulářů. HTML tagu <a> se nastaví třída „ajax“ a pro vyvolání událostí se využije signálů *„handle“*.

```
<a href="{link smaz!, id}"  class="ajax">smaž</a>
```

Signál se spustí na pozadí na způsob vláken a provede například nějaký databázový příkaz. V tomto případě smaže záznam z databáze.

```
 public function handleSmaz($id) {
      Model::smaz($id);
      if(!$this->isAJAX())
           $this->redirect('this');
      else
           $this->invalidateControl('vypis');

}
```

3 Reimplementace utilit

Hlavním důvodem reimplementace celé webové části Virtlabu je ten, že postupným rozvíjením systému, webová část narostla do takového stavu, kdy je problematické do kódu vložit jakoukoli novou složitější funkcionalitu. Stará verze také postrádá výhody nových technologii a nerespektuje principy objektově orientovaného programování což je u tak rozsáhlého projektu výrazný nedostatek.

Z tohoto důvodu bylo rozhodnuto, že bude systém reimplementován. Martin Grames ve své diplomové práci pro nový Virtlab vybral skriptovací jazyk PHP, který byl použit také u předchozí verze. Ovšem pro novou verzi vybral framework Nette, který disponuje většinou nových technologii a samozřejmě využívá všechny výhody Objektově orientovaného programování. [1]

3.1 Modul pro hlášení chyb

Modul umožňuje uživatelům nahlásit vzniklé chyby, které se vyskytnou během používání systému. Tyto chybové zprávy se dále odesílají do systému pro sběr chyb http://virtis.viakis.net [2].

VirtIS je jednoduchý informační systém pro Virtlab sloužící k řízení a plánovaní vývoje. Zajišťuje evidenci jednotlivých úloh, testů a časových fází. Systém VirtIS má nastarost Kateřina Bambušková, která se také stará o chod Virtlabu.

3.1.1 Analýza stávajícího systému

Celý modul pro hlášení chyb je v podstatě jeden formulář obsahující základní údaje o vniklé chybě. Formulář obsahuje pole pro zadávání data a času a krátký popis chyby. Popis sám o sobě nemusí někomu dávat smysl, proto se ve formuláři nabízí možnost vložit obrázek, nebo spíše PrintScreen, který usnadní pochopení této vzniklé chyby. Dále se pak formulář zpracuje a pomocí HTTP hlavičky se data odesílají na VirtIS. V hlavičce se kromě obsahu formuláře odesílají skryté data o přihlášené uživateli a místu odkud byla chyba odeslána. Místem je myšlena lokalita a URL adresa. VirtIS tyto data dále zpracuje a zařadí je do kategorie BUGS.

3.1.2 Návrh řešení

Tak aby bylo zachováno pořadí a rozmístěni jednotlivých utilit, bude potřeba pro tento modul vytvořit novou kategorii v menu, to také znamená vytvoření nového presenteru. Modul by měl být jednoduchý a pro všechny uživatele srozumitelný. Pro potřeby formuláře bude využita třída *form*, která umožňuje nastavit formuláří parametr *action* a tak odeslat data HTTP hlavičkou na vzdálený sever. Po odeslání těchto dat již není potřeba nic jiného provádět. Systém VirtIS si data sám zpracuje. Stačí jen uživateli zobrazit zprávu o tom, že byla zpráva odeslána.

3.1.3 Implementace

Pro tento modul byl vytvořen presenter s názvem *WriteUsPresenter.php.* Presenter obsahuje pouze komponentu *createComponentBugsForm* pro vytvoření a ošetření formuláře a jednu funkci *virtisOk* pro potvrzení při odeslání.

Samotná implementace komponenty spočívá v nastavení *setAction* pro odeslání formuláře na již zmíněný systém VirtIS. Ten očekává hodnoty ve správném tvaru tak, aby je mohl dále zpracovat. Po zjištění funkčnosti při dodržení stejných hodnot odeslaných z nového systému Virtlab nebylo potřeba dále zjišťovat, jak VirtIS data zpracovává.

VirtlabReferer – podle tohoto parametru obsahující URL, VirtIS zjistí, jestli byla zpráva odeslána z Virtlabu a tedy jestli ji má dále zpracovat
sUser – obsahuje login, jméno a příjmení přihlášeného uživatele, který zprávu odeslal
sEmail – e-mail přihlášeného uživatele
sLokality – lokalita odkud je zpráva odeslána

Krom těchto parametrů jsou dále odesílány samostatná data formuláře:

sUserTime – datum a čas výskytu chyby
pkFile – PrintScreen chyby
sBugDescription – popis vystihující vzniklou chybu

3.1.4 Testování

Několikanásobným odesláním různě vyplněného formuláře byla otestována funkčnost modulu pro hlášení chyb. Všechny zprávy v systému VirtIS jsou zobrazeny správně. Dále jsem si zajistil přístup do systému tak, abych mohl zkontrolovat funkčnost zprávy také z pohledu administrátora.

3.2 Modul pro sledování uživatelů

Tento modul je pouze jedna tabulka zobrazující data z databáze o přihlášených uživatelích, kteří se aktivně nachází na Virtlabu. Tak má kterýkoli uživatel možnost vidět ostatní aktivní uživatele.

3.2.1 Analýza stávajícího systému

Modul pro sledování uživatelů pracuje nad jedinou tabulkou databáze a to tabulkou *who*. Zde se zapisují aktuální data o přihlášených uživatelích. Jejich login, identifikátor stránky, a čas poslední aktivity. V dřívější verzi systému to uskutečňoval databázový příkaz *REPLACE*, který se volal v *index.php,* tzn. při každém kliknutí uživatele, byla data aktualizována.

3.2.2 Návrh implementace

Tabulka modulu zobrazující aktivní uživatele na Virtlabu by měla obsahovat všechny informace, které byly součástí tabulky starého systému. Dále musí být přehledná a samozřejmě nezobrazovat uživatele, kteří již aktivní nejsou. To zajistí podmínka v SQL příkazu zobrazující jen uživatele aktivní méně jak 20 minut. Zachycení aktivity uživatele by mělo být umístěno na správném místě, tak aby se při každém obnovení stránky data aktualizovala.

3.2.3 Implementace

Při implementaci do nové verze systému nastal problém se zapisováním místa, na kterém se dotyčný nachází. Dříve to řešil identifikátor každé stránky, který se zapisoval do databáze. Z nástupem Nette přesměrovávající své stránky na tzv. „hezké url“, které je bezpochyby

nezbytnou součástí nových webu, naplňujících nároky SEO, vznikla otázka. Co zapsat do databáze namísto identifikátoru stránky? Z tohoto důvodu se pozměnila databázová tabulka.

	Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra	Akce
☐	user_id	varchar(20)	utf8_unicode_ci		Ne	*Žádná*		
☐	page_label	varchar(200)	utf8_unicode_ci		Ne	*Žádná*		
☐	page_url	varchar(200)	utf8_unicode_ci		Ne	*Žádná*		
☐	time	datetime			Ne	*Žádná*		

Obrázek 2: Struktura databázové tabulky who

user_id – login uživatele

page_label – název stránky, na které se uživatel nachází např. („Active user“)

page_url – url té stránky např. („/new/user-admin/active-user/“)

time – datum a čas poslední aktivity uživatele

Aby bylo možné zapisovat všechna data do databáze, bylo potřeba přidat do třídy *Navigation* funkci, která vrátí jak url, tak i název stránky.

```
public function findByLinkLabel($link){
	foreach ($this->root->getComponents(TRUE) as $item) {
		if ($item->url === $link) {
			$pole = array('label'=>$item->label,'url'=>$item->url);
			return $pole;
		}
	}
	return NULL;
}
```

Příkaz *REPLACE,* zapisující data do databáze, je naimplementován v *BasePresenter.php* na konci komponenty *Navigation.* Zde má tento příkaz veškerá potřebná data pro zápis a je spouštěn při každém načtení stránky přihlášeného uživatele.

```
if($user->isAuthenticated() && (!isset($_SESSION['man-oprava']) ||
$_SESSION['man-oprava'])) {
      $pole = $navigation->findByLinkLabel($this->link($this-
>getAction()));
      BaseModel::whoReplace( $user->identity->getName(),
      $pole['label'], $pole['url']);
} else
      $_SESSION['man-oprava'] = true;
```

Protože se příkaz spouští při každém obnovení stránky, ale ne vždy je tento příkaz potřeba, tak je vytvořen session zabraňující spuštění funkce. To se zejména hodí při ošetřování formulářů, jejichž podmínky čerpají data z jiné databáze, než využívá tato funkce.

Samotný výpis pak obstarává SQL příkaz omezený na výpis uživatelů aktivních méně jak 20 minut od aktuálního času. Zobrazí veškerá data z tabulky *who.*

```
dibi::query('SELECT user_id, page_label, page_url,(UNIX_TIMESTAMP(NOW()) -
UNIX_TIMESTAMP(time)) as last_act FROM [who] WHERE time> TIMESTAMPADD( HOUR ,
-1, NOW( ) ) ORDER BY last_act ASC');
```

`UNIX_TIMESTAMP() -` vrátí rozdíl data od 1.1.1970

3.2.4 Testování

Pro testování bylo vytvořeno několik přihlašovacích údajů. Na ty jsem se pak průběžně přihlašoval na 2 počítačích, na každém z nich jsem využil dvou různých prohlížečů. Postupným sledováním jsem pak dospěl ke správné funkčnosti modulu. Modul dle očekávání pracoval správně.

3.3 Graf zátěže

Graf zátěže slouží jako grafické nástroj pro zobrazení rezervací z databáze. Uživatel tak má možnost vidět na časové ose využití všech laboratorních zařízení. Tyto zařízení jsou uspořádány pod sebou. Každá hodina, kdy zařízení má rezervaci, je zvýrazněna.

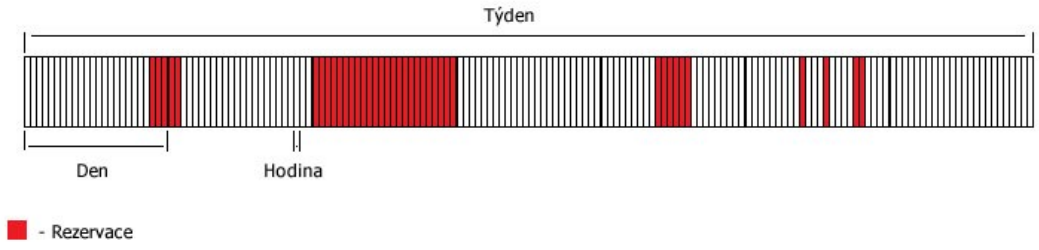
3.3.1 Analýza stávajícího systému

Zpracování grafu obstarává formulář obsahující dva parametry, datum začátku a konce. V tomto rozmezí se má graf vykreslit. Rozmezí je dále omezeno na jeden měsíc. Třída *Graph* starající se o správné vykreslení grafu pracuje s frameworkem PEAR pro verzi PHP 4. Což je také zásadní problém při implementaci do nového projektu, který využívá PHP verze 5.3. Graf se v PEAR balíčku vykreslí a převede na obrázek. Ten se pak zobrazí uživateli. Načítání grafu, vykresleném jako obrázek, je poměrně časově náročné.

3.3.2 Implementace

Jako první nápad na řešení bylo graf reimplementovat z původní verze Virtlabu pomocí balíčku PEAR, ale nedokázal jsem se vypořádat s rozdíly mezi verzemi PHP. Dalším nápadem bylo najít graf na internetu s volnou licencí, odpovídající původnímu grafu. Ovšem po dlouhém hledání na internetu, nebyl graf k nalezení. Nakonec jsem se rozhodl vytvořit vlastní graf, který bude odpovídat původnímu grafu zátěže.

Graf je na první pohled jedna velká tabulka, kde řádky jsou jednotlivé laboratorní zařízení a sloupce dny v rozmezí hodnot zadaných do formuláře.



Obrázek 3: Rozložení jednoho řádku grafu

Graf se nachází v kategorii rezervací, proto nebylo potřeba vytvářet nový presenter. Implementace se nachází v již vytvořeném presenteru *RezervationPresenter.php*, ale data čerpá z databáze dvl-reser v modelu *Managamet.php*, který má implementovaný přístup do této databáze. Pro vykreslení grafů je potřeba napsat správně SQL příkaz tak, aby vracel asociativní pole ve tvaru:

Array[zarizeni][pocet_hodin_od@pocet_hodin_do]

```
return dibi::query('SELECT rd.device,
CONCAT((TIME_TO_SEC(TIMEDIFF(%t,r.from_time))/60/60),"@",
(TIME_TO_SEC(TIMEDIFF(%t,r.to_time))/60/60)) hodin
FROM reservations r, reserved_devices rd
WHERE r.resid=rd.resid AND
 (r.from_time >= %t AND r.to_time <= %t)
ORDER BY device DESC
',$prvni,$prvni,$prvni,$posledni)->fetchAssoc('device,hodin');
```

Funkce použité v SQL přikazu:

- **CONCAT(h1,'@',h2) -** sloučí „h1" a „h2" a mezi tyto řetězce vloží oddělovač v mém případě zavináč @
- **TIME_TO_SEC(d)** – převede datum „d" na sekundy
- **TIMEDIFF(d1,d2)** - vrátí rozdíl mezi daty „d1" a „d2" ve formátu „00:00:00.000000"

Teď má presenter potřebné hodnoty a to od jaké hodiny se má konkrétní rezervace v tabulce vykreslit a kdy má skončit. Každá hodnota je oddělena zavináčem, pomocí něhož je pak řetězec rozdělen. Příkaz omezuje parametry zadané do formuláře, aby nenačítal zbytečně více informací než je potřeba.

Několik vnořených cyklu *foreach* se postará o vykreslení. Graf musí být ovšem vykreslen od začátku do konce. O to se postará ještě jeden cyklus *for* začínající na nulté hodině a končící poslední hodinou.

```
$table = Html::el('table');
foreach ($graf_zarizeni as $device => $orders) {
	$tr = $table->create('tr');
	$tr->create('th')->setText($device);
	for($i=0;$i<$this->rozdil_data($prvniDen, $posledniDen);$i++) {
		foreach ($orders as $od => $order) {
			list($from,$to) = explode("@", $od);
			if( ((ceil($from*(-1)) < $i &&  ceil($to*(-1)) > $i) ||
			((($to*(-1)) - ($from*(-1))) < 1) &&
			ceil($from*(-1)) == $i))) {
				$co = 'false';
				break;
			} else $co = 'true';
		}
		Pocitadlo pro vypis grafu
	}
}
```

$prvniDen – počáteční datum zadaný do formuláře
$posledniDen – konečný datum zadaný do formuláře
rozdil_data() – vrátí rozdíl mezi daty v hodinách
$graf_zarizeni – objekt DibiRow s hodnotami z databáze

Aby vykreslování nebylo zbytečně náročné a zdlouhavé byly využity výhody AJAXu. Ten se postará na pozadí o vykreslení grafu. Uživatel tak nevidí postupné vykreslování, jen ikonu upozorňující načítání.

Obrázek 4: Ikona načítání grafu

V implementaci je stále jedna nevýhoda. Graf je složen z HTML prvků <table>. Pro malý rozsah mezi daty a pár zařízení je to v pořádku. Ovšem pokud jde o více rezervací například:

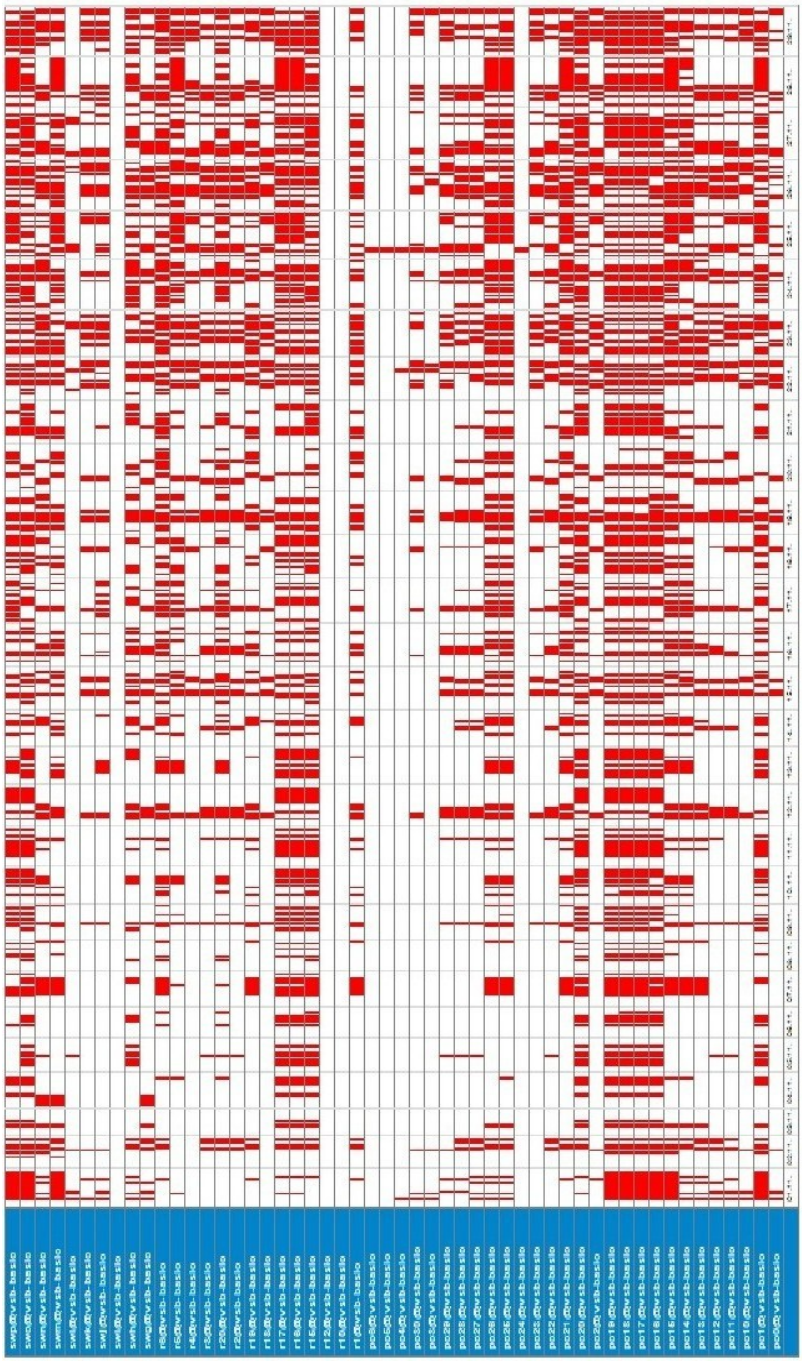
20 zařízení v rozsahu 1 měsíc - $(31dní * 24hodin) * 20zařízení = 14788$

Tabulka tak dosahuje velkého počtu HTML prvků, což má také vliv na rychlost načítání. Z toho důvodu je implementováno na výpis také počítadlo, které nastaví na nerezervované políčka atribut „*colspan*“. Ten je sloučí dohromady a tím ušetří počet HTML prvky a tak zrychlí načítání. Počítadlo také zamezí tomu, aby *colspan* přesáhl hodnotu 24 (hodin), to by se graf stával nepřehledným.

```
$pole[$b] = $co; // naplní pole hodnotami (rezervace/bezrezervace)
$a = 0;

if($b == 24) { // pokud je 1 den -> 24 hodin
	if(!in_array('false', $pole)) // pokud v tento den nejsou rezervace
		$tr->create('td colspan=24 class="l"');
	else
		for($j=1;$j<=24;$j++) // vypiš 24 hodin
			if($pole[$j] == 'true') // pokud není rezervace
				// Pokud není rezervace ani na následující hodinu
				if($j != 24 && $pole[$j+1] == 'true')
					$a++;
				else
					if($a == 0) {
						if($j==24) $l = ' class=l'; else $l='';
						$tr->create('td'.$l);
					} else {
						if($j==24) $l = ' class=l'; else $l='';
						$a++;
						$tr->create('td colspan='.$a.' '.$l);
					}
			else { // pokud je rezervace
				if($j==24) $l = ' l'; else $l='';
				$tr->create('td class="a'.$l.'"');
				$a = 0;}
			$pole = array(); $b=1; // vynuluj hodnoty pro dalsi den
} else $b++;
```

Tento způsob implementace grafu má velkou výhodu oproti předchozí verzi, vykreslené jako obrázek. Graf lze totiž přizpůsobit dle potřeby. Poněvadž je složen z HTML prvků dá se na graf navázat jakýkoliv vzhled pomocí CSS stylů nebo lze využít různé *JavaScriptové* funkce.



Obrázek 5: Nový graf zátěže

3.3.3 Testování

Testování grafu zátěže se dá rozdělit na 2 části:

1. Časová náročnost
    - Postupným testováním a následnými upravami bylo docíleno poměrně slušné časové náročnosti grafu. Graf se v období velké zatíženosti vykresluje zhruba 4s, což je oproti předchozí verzi dobré.
2. Správné zobrazení grafu
    - Pro testování grafu jsem naplnil databázi hodnotami ze staré verze Virtlabu. Poté jsem vizuálně porovnal vykreslení hodnot v obou grafech. Po pár úpravách bylo docíleno shody mezi grafy, což také bylo cílem testování.

4 Správa rezervačního serveru

4.1 Analýza

Rezervační server posílá seznam dostupných zařízení ve formátu XML každé lokalitě, na základě časového rozvrhu a dalších informací obsažených v databázi *dvl-reser*. Do této databáze si ukládá také data o úspěšně provedených rezervacích. Ovšem časové rozvrhy a seznam odstavených zařízení do databáze musí vkládat administrátor sám.

Dříve administrátoři vkládali tyto informace přímo do databáze, což není moc efektivní způsob správy tohoto serveru. U mého návrhu už by tomu tak být nemělo. Nový systém by měl totiž osahovat také uživatelské rozhraní pro správu rezervačního serveru.

Databáze *dvl-reset* obsahuje sedm tabulek. Ty jsou rozděleny do 3 základních skupin: odstavené zařízení, časové rozvrhy a rezervace zařízení.

4.1.1 Odstavené zařízení

Obsahuje 2 tabulky:

decommissioned_devices

	Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra	Akce
☐	device	varchar(20)	utf8_czech_ci		Ne			
☐	decommission_id	int(11)			Ne	*Žádná*		

Obrázek 6: Struktura databázové tabulky decommissioned_device

decommissions

	Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra	Akce
☐	decommission_id	int(11)			Ne	*Žádná*	AUTO_INCREMENT	
☐	from_time	datetime			Ne	0000-00-00 00:00:00		
☐	to_time	datetime			Ne	0000-00-00 00:00:00		
☐	locality	varchar(23)	utf8_czech_ci		Ne			
☐	user_group	text	utf8_czech_ci		Ne	*Žádná*		

Obrázek 7: Struktura databázové tabulky decommissions

Všechna zařízení obsažena v tabulce *decommissioned_devices* jsou odstavena z provozu, což znamená, že je rezervační server nenabízí k použití. Datum, čas, pro jakou skupinu a lokalitu má být zařízení odstaveno se dále nastavuje v tabulce *decommissions*. Tabulky jsou ve vztahu (*decommissioned_devices, decommissions)* N:1 takže pro jeden řádek v tabulce *decommissions* může existovat více zařízení. Sloupec *device* má tvar „id_zařízení@lokalita“, to platí také pro všechny později zmíněné tabulky se sloupcem *device*.

4.1.2 Časové rozvrhy

Obsahují 3 tabulky:

devices_in_timetable

	Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra	Akce
☐	device	varchar(20)	utf8_czech_ci		Ne			
☐	timetable_id	varchar(23)	utf8_czech_ci		Ne			

Obrázek 8: Struktura databázové tabulky device_in_timetable

entries

	Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra	Akce
☐	timetable_id	varchar(23)	utf8_czech_ci		Ne			
☐	day	tinyint(4)			Ne	0		
☐	from_hour	tinyint(4)			Ne	0		
☐	from_minute	tinyint(4)			Ne	0		
☐	to_hour	tinyint(4)			Ne	0		
☐	to_minute	tinyint(4)			Ne	0		

Obrázek 9: Struktura databázové tabulky entries

timetables

	Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra	Akce
☐	timetable_id	varchar(23)	utf8_czech_ci		Ne			
☐	from_time	datetime			Ne	0000-00-00 00:00:00		
☐	to_time	datetime			Ne	0000-00-00 00:00:00		
☐	locality	varchar(23)	utf8_czech_ci		Ne			
☐	user_group	text	utf8_czech_ci		Ne	*Žádná*		

Obrázek 10: Struktura databázové tabulky timetable

Časové rozvrhy slouží pro nastavení jednotlivých lokalit. Nastavují se většinou na delší období například na jeden semestr nebo školní rok. V tomto období se dále, pro určitou lokalitu, nastavuje týdenní rozvrh, v kterém si může lokalita rezervovat zařízení. K tomu slouží tabulky

(*timetables, entries*), které jsou v závislosti 1:N. V třetí tabulce (*devices_in_timetable*) se opět nastavují zařízení dostupné pro každou lokalitu a uživatelskou skupinu.

4.1.3 Rezervované zařízení

Obsahuje 2 tabulky:

reservations

	Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra	Akce
☐	resid	varchar(23)	utf8_czech_ci		Ne			
☐	from_time	datetime			Ne	0000-00-00 00:00:00		
☐	to_time	datetime			Ne	0000-00-00 00:00:00		

Obrázek 11: Struktura databázové tabulky reservations

reserved_devices

	Sloupec	Typ	Porovnávání	Vlastnosti	Nulový	Výchozí	Extra	Akce
☐	device	varchar(20)	utf8_czech_ci		Ne			
☐	resid	varchar(23)	utf8_czech_ci		Ne			

Obrázek 12: Struktura databázové tabulky reserved_devices

Do těchto tabulek si rezervační server sám vytváří záznamy o úspěšných rezervacích. Tabulky jsou ve vztahu (*reserved_devices, reservations*) N:1. Ukládá se do nich čas rezervace a pro jaké zařízení je rezervace vytvořena. Stejně jako *device,* má také sloupec *resid* tvar „id@lokalita“, kdy *id* musí odpovídat *id* s tabulky *reservations* v databázi *dvl-www.*

4.2 Návrh implementace

Jelikož tento návrh má pracovat nad již připravenou funkční databází rezervačního serveru, není potřeba se dále při implementaci zaobírat strukturou databáze. Jak již bylo zmíněno v předchozích kapitolách, rozhraní by mělo být rozděleno na 3 základní části.

Dále by každá z těchto částí měla obsahovat jak formuláře pro ukládání dat, tak i tabulky pro výpis již uložených dat z databáze. Neměla by tam samozřejmě chybět možnost editace a smazání původních nebo nově zapsaných záznamů. Konečná podoba rozhraní by měla být přehledná a jednoduchá na používání.

4.3 Implementace

V projektu je pro toho rozhraní vytvořený nový presenter *ManagamentPresenter.php,* který bude obstarávat veškerou logiku pro všechny 3 části správy rezervačního serveru. Ovšem presenter by neměl, podle modelu MVC, mít přímý přístup do databáze. Z toho důvodu bylo potřeba vytvořit také nový model *Managament.php*, který má přístup k databázi rezervačního serveru *dvl-reser*.

V novém systému je pro přístup k databázi webové části *dvl-www* naimplementovaný ORM modul Ormion. Pro rozsáhlý přístup k databázi je modul ORM přínosem. Složitá implementace se vyrovná s pozdějším využíváním modulu, kdy místo neustálých psaní SQL příkazů se využijí naimplementované funkce. Ovšem pro potřeby rezervačního serveru, který obsahuje pouze 7 tabulek, je složitá implementace modulu ORM zbytečná. V nově vytvořeném modelu *Managament.php* je využita mini databázová vrstva dibi. Stále ovšem jako hlavní databáze systému zůstává *dvl-www*. Její přístup je definován přímo v souboru *config.ini.*

Připojení k databázi *dvl-reser* je vloženo do souboru *bootstrap.php* a pojmenováno jako *reser*.

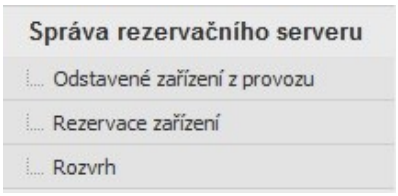
```
$options = array(

    'driver'   => 'mysql',
    'host'     => '127.0.0.1',
    'username' => 'user',
    'password' => 'password',
    'database' => 'dvldb-reser',
);
dibi::connect($options, 'reser');
```

Toto pojmenování umožní přepínání mezí přístupy do různých databází. V třídě Managament je pak statická funkce *connect*, která nastaví připojení k pojmenované databázi *reser*.

```
public static function connect() {
      return dibi::activate('reser');
}
```

Bylo také zapotřebí vytvořit novou kategorii v menu obsahující jen části tohoto rozhraní tak, aby bylo odděleno od ostatních položek menu. To umožňuje skrýt celou kategorii před běžnými uživateli, kteří nemají k těmto administrátorským rozhraním přístup. Práva pro jednotlivé uživatelské skupiny a jejich role v systému se nastavují v souboru *ACL.php*.



Obrázek 13: Menu správy rezervačního serveru

Dále pro výpisy z databáze potřebné ve všech třech částech rozhraní, se využil doplněk *DataGrid*. Ten již v projektu byl zakomponován. Bylo by zbytečné hledat nebo implementovat novou utilitu pro takový výpis dat, pokud je již dostupná v projektu.

Tento doplněk frameworku Nette slouží, jak již bylo zmíněno, pro výpisy dat. Jeho hlavní výhodou je možnost data rychle třídit, filtrovat nebo s daty manipulovat. Na jednotlivé řádky výpisu se dají nastavit různé akce pro manipulace s daty. Pro mé účely byly využity pouze možnosti záznam smazat, editovat nebo zobrazit detail záznamů

Obrázek 14: DataGrid doplněk

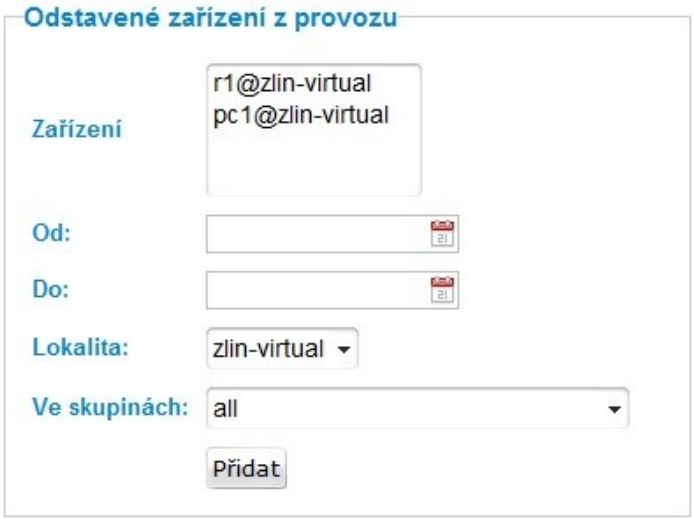
Všechny formuláře správy rezervačního serveru dále potřebují znát zařízení, která mu poskytuje rezervační server. Ty jsou, jak již bylo řečeno, posílány v XML souboru. Pro načítání zařízení ze souboru byla vytvořena samostatná třída *Device*. Ta se nachází v souboru *Device.php* a obsahuje následující funkce:

getDevices() – vrátí všechna zařízení obsaženy v XML souboru v indexovaném poli

getDeviceValue($key) - vyhledá a vrátí název zařízení podle jeho identifikátoru, hodnotu vrací jako řetězec (string)

4.3.1 Odstavené zařízení

Zařízení laboratoře není bezchybné a mnohdy se může porouchat nebo mít jiný problém, který by zabránil zařízení používat. Pak by mohly vznikat nevysvětlitelné chyby v programu. A tak by uživatelé rezervující si určitý čas, ze své kvóty, nemuseli být zrovna spokojení. S toho důvodu také existují tyto dvě tabulky, zabraňují rezervačnímu serveru nenabízet zařízení dočasně odstavené z provozu.

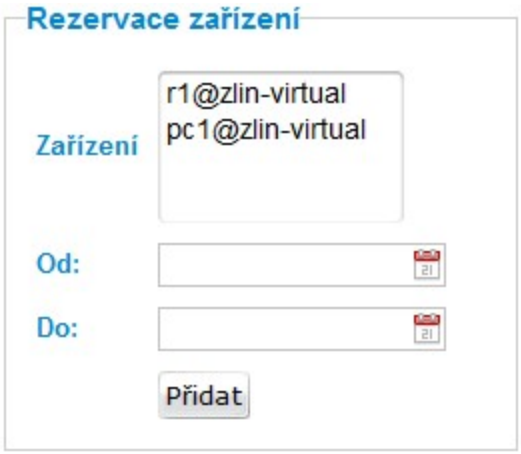


Obrázek 15: Formulář odstavených zařízení

Na obrázku je vidět formulář pro odstavení laboratorního zařízení. Obsahuje samozřejmě možnost vybrat zařízení, které má být odstaveno. Zařízení nebude odstaveno navždy, proto se také nastavuje časové rozmezí. V tomto rozmezí nebude zařízení dostupné pro lokality a uživatelské skupiny, které se nastavují jako poslední dvě položky formuláře.

4.3.2 Rezervované zařízení

Zde má administrátor možnost rezervovat zařízení samostatně mimo zadanou úlohu. K tomu slouží jednoduchý formulář obsahující pouze jméno zařízení a časové rozmezí, v kterém má být zařízení rezervováno. Odesláním formuláře se vloží záznam do databáze, ovšem identifikátor rezervace je potřeba převzít z tabulky *reservations* nacházející se v databázi *dvl-www* tak, aby záznamy byly v korektním stavu. Dva stejné záznamy jsou ve dvou různých databázích což má za následek to, že není možné na tabulky nastavit cizí klíče. Z toho důvodu záznam musí být vložen do obou zmíněných tabulek a zamezit tak nekorektním záznamům.

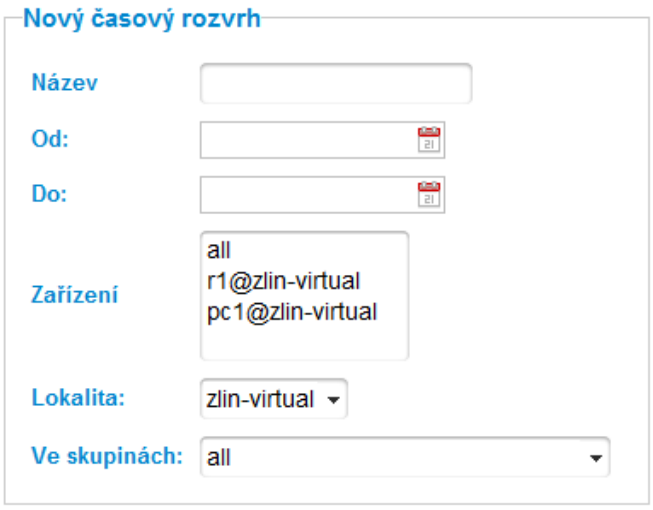


Obrázek 16: Formulář rezervovaných zařízení

4.3.3 Časové rozvrhy

Časové rozvrhy slouží jako omezení rezervací pro různé lokality nebo uživatelské skupiny. Vytvoření takového rozvrhu obstarává formulář skládající se ze dvou částí. První část formuláře obstarává nastavení pro rozvrh a druhá část pak časový plán pro toto nastavení.

V první části formuláře musí být nastaven *Název* časového rozvrhu, který také slouží jako jedinečný identifikátor. Dalšími dvěma položkami se nastavuje časové rozmezí platnosti rozvrhu. To většinou odpovídá délce jednoho semestru nebo školního roku. Není to však podmínkou. Dále lze nastavit pro jakou lokalitu a uživatelskou skupinu, bude rozvrh vytvořen. Nakonec je zde možnost nastavit, která zařízení bude mít uživatelská skupina k dispozici v časovém plánu definovaném v druhé části formuláře. Navíc zde přibyla možnost zvolit jako zařízení položku *all*. Ta označuje, že skupina má k dispozici všechna zařízení.



Obrázek 17: První část formuláře časových rozvrhů

Druhá část formuláře pak nastavuje týdenní plán pro nastavení časového rozvrhu nastaveného v první části formuláře. Nastavují se zde časové údaje v hodinách a minutách pro každý den v týdnu zvlášť.

Obrázek 18: První část formuláře časových rozvrhů

4.4 Testování

Cílem testování správy rezervačního serveru je správný zápis hodnot do databáze a následný výpis. Všechny hodnoty se do databáze zapisují správně ve správném pořadí a tvaru. To lze také ověřit na výpisu pomocí DataGridu hned pod každým formulářem. Každou s části správy rezervačního serveru jsem odzkoušel pro různé prohlížeče. Při testování byl vyplněn každý formulář nejméně 20 krát, dle rozsahu formuláře, různými hodnotami. Po odladění chyb bylo docíleno správné funkčnosti a ošetření vstupu formulářů všech již zmíněných částí.

5 Závěr

Zadáním této práce bylo reimplementovat vybrané utility a implementovat novou správu rezervačního systému do nové verze Distribuované virtuální laboratoře počítačových sítí.

Cílem správy rezervačního serveru je poskytnou administrátorům Virtlabu přijatelnější možnost spravovat rezervační server za pomocí uživatelského rozhraní. Tak aby rozhraní bylo dosti přehledné, bylo rozděleno na 3 základní části: rezervace zařízení, odstavené zařízení a časové rozvrhy. Všechny části obsahují formuláře pro přidání nových záznamů do databáze a následně editaci těchto záznamů. Dále obsahují doplněk DataGrid pro výpis a možnost smazání. Rozhraní bylo důkladně otestováno a nyní je vše připraveno na ostrý provoz.

Reimplementace vybraných utilit je jen pokračováním v robusní reimplementaci celého systému do nové podoby. Modul pro sledování aktivních uživatelů a modul pro hlášení chyb se úspěšně povedlo přepsat do nového systému. Graf zátěže je vytvořen úplně nový, z důvodu nekompatibility odlišných verzí PHP.

Nad rámec zadání byl systém dále rozšířen o doplňky frameworku Nette. DataTimePicker pro vykreslení formulářových prvků obsahující časové údaje, který je uživatelsky přívětivější než obyčejný input formuláře. Druhé rozšíření pojednává o využití AJAXových funkcí při odesílání událostí formulářů nebo hypertextových odkazů. Rozepsáno je i nastavení a využití těchto rozšíření.

Všechny zmíněné utility a moduly jsou implementovány ve frameworku Nette, který pro nový systém vybral Martin Grames jako téma své diplomové práce. Dále je pro přístup do databáze využit modul ORM Ormion a mini databázová vrstva dibi. Mé znalosti byly před započetím práce omezeny pouze na jazyk PHP a MySQL. Myslím si, že práce byla pro mě velkým přínosem a věřím, že získané znalosti využiji ve svém budoucím zaměstnání.

Literatura a informační zdroje

[1] GRAMES, Martin.*Reimplementace řídící aplikace systému Virtlab s použitím moderních webových technologií.* Ostrava, 2010. Diplomová práce. VŠB-TU Ostrava, FEI.

[2] *Nette Framework: Dokumentace* [online]. Nette Foundation, c2010. Dostupne z WWW: <http://nettephp.com/cs/dokumentace>

[3] NOVÁK, Radek. *Robustní reimplementace prototypového rešení serverových komponent systému Virtlab.* Ostrava, 2010. Diplomová práce. VŠB-TU Ostrava, FEI.

[4] Stránky projektu Virtlab [online]. Virtuální laboratoř počítačových sítí. Dostupné z WWW: http://www.virtlab.cz

[5] GILMORE, Jason W. Velká kniha PHP a MySQL 5: kompendium znalostí pro začátečníky i profesionály. Vyd. 1. [i.e. 2. vyd.]. Brno: Zoner Press, 2007, 864 s. ISBN 80-86815-53-6.

[6] Dibi : Database Abstraction Library for PHP 5 [online].Nette Foundation, c2010. Dostupné zWWW: <http://dibiphp.com>.

[7] GRUDL, David. Začínáme s Nette Framework. Zdroják : tvorba webových stránek a aplikací [online]. c2009. Dostupný z WWW: <http://zdrojak.root.cz/serialy/zaciname-s-nette-framework/>.

Přílohy

Příloha A

Obsah doprovodného CD

1. Readme.txt - soubor s popisem datového CD.
2. Virtlab – adresář s projektem Virtlabu.
3. Text - bakalářská práce ve formátu PDF.

Seznam obrázků